Managing Kernel Extensions

Session 108

Managing Kernel Extensions

Craig Keithley
USB and FireWire Technology Evangelist

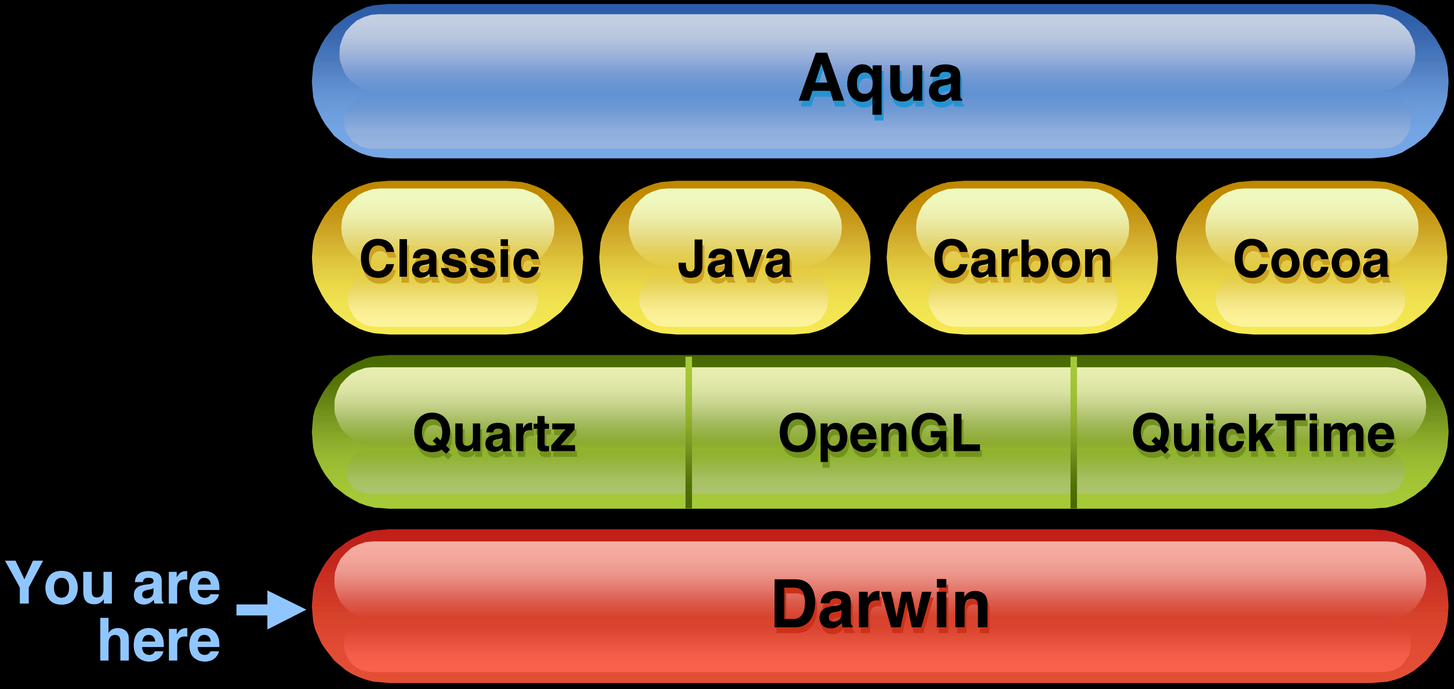
Managing Kernel Extensions

Dean Reece
I/O Kit Team Manager

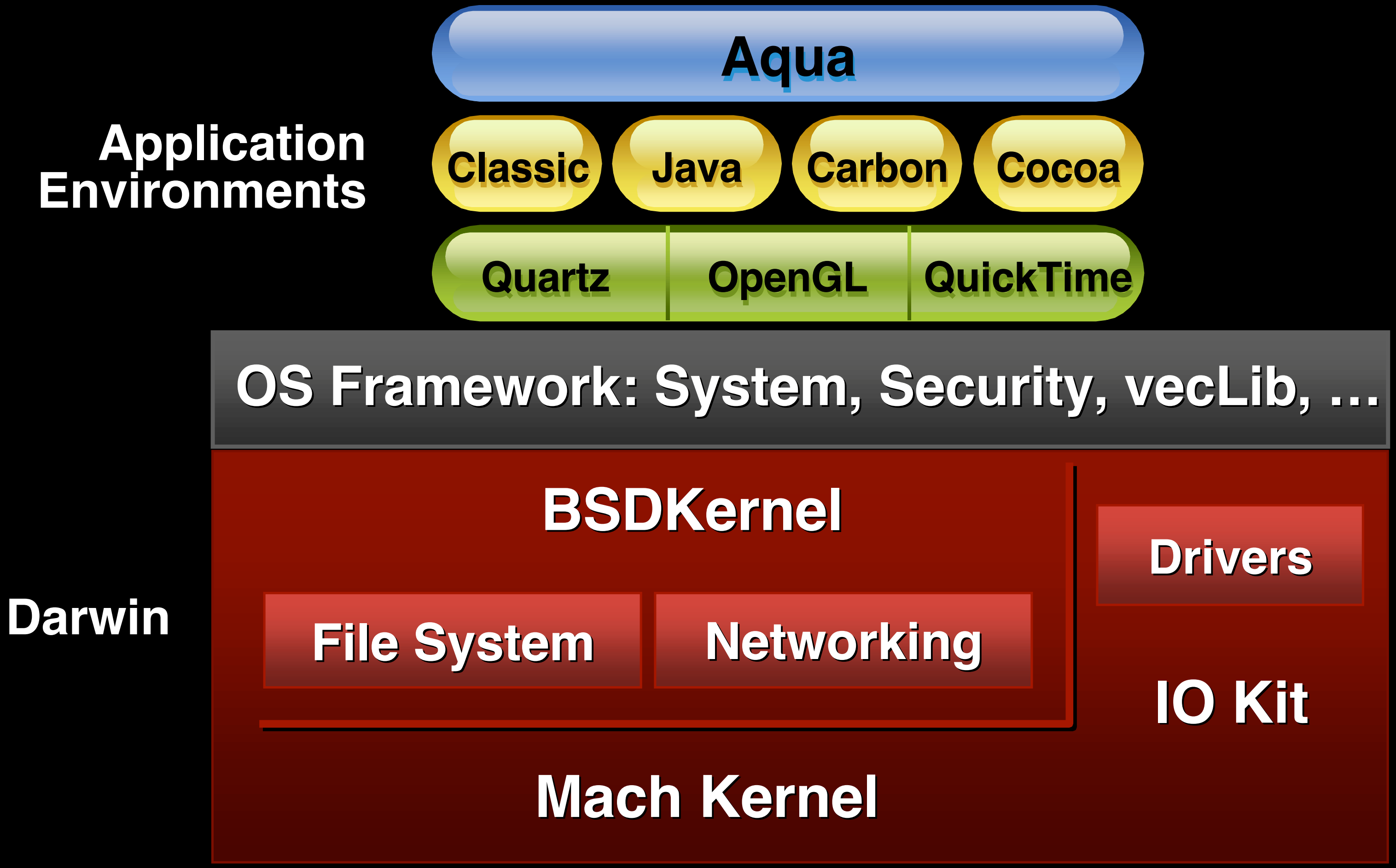
What You Will Learn

- What KEXTs are
- When (not) to use them
- How KEXTs are managed, developed and distributed
- Pitfalls to avoid

Mac OS X Architecture



Darwin/Core OS



Mac OS X Kernel Extensions

- Kernel Extensions (KEXTs) add functionality to the Mac OS X Kernel
- There are 3 types of KEXTs
    - I/O Kit Drivers and Families
    - Network Kernel Extensions (NKEs)
    - Filesystem Extensions

Appropriate Uses of KEXTs

- Do not use a KEXT unless absolutely necessary:
  - Development in the kernel is harder
  - In-kernel resources are more expensive
  - Crashes are fatal
- A KEXT is only necessary if
  - You must respond to an interrupt
  - Your primary client is in the kernel

Anatomy of a KEXT

- A KEXT is a bundle with ".kext" extension
  - An indivisible item to users
  - A folder of resources to developers
- May contain
  - A descriptive property list (required)
  - A binary to load into the kernel
  - Localizable strings files, icons
  - Utility binaries, firmware images
  - Other bundles (including KEXTs)

KEXT Example: Structure

- IOUSBFamily.kext
    - Contents
        - Info.plist
        - MacOS
            - IOUSBFamily
        - PlugIns
            - AppleUSBComposite.kext
            - IOUSBLib.bundle
            - IOUSBUserClient.kext
            - ...

KEXT Example: Structure

- IOUSBFamily.kext
  - Contents
    - Info.plist
    - MacOS
      - IOUSBFamily
    - PlugIns
      - AppleUSBComposite.kext
      - IOUSBUserClient.kext
      - IOUSBLib.bundle
      - ...

KEXT Example: Info.plist

- Info.plist is an XML property list that identifies the KEXT and its contents
- Critical properties:
    - CFBundleIdentifier = “com.you.driver”
    - CFBundleVersion = “1.2.3b4”
    - CFBundleShortVersionString = “1.2.3”
    - OSBundleLibraries = “com.apple.io…”
    - IOKitPersonalities = { … }

KEXT Example: Structure

- 🗁 IOUSBFamily.kext
  - 🗁 Contents
    - 🗎 Info.plist
    - 🗁 MacOS
      - 🗎 **IOUSBFamily**
    - 🗁 PlugIns
      - 🗁 AppleUSBComposite.kext
      - 🗁 IOUSBUserClient.kext
      - 🗁 IOUSBLib.bundle
      - ...

KEXT Example: Binary

- Optional
- Mach-O format
- There are two standard entry points
    - **module_start()**
    - **module_stop()**
- I/O Kit KEXTs do not use module_start/stop

KEXT Example: Structure

- IOUSBFamily.kext
  - Contents
    - Info.plist
    - MacOS
      - IOUSBFamily
    - PlugIns
      - AppleUSBComposite.kext
      - IOUSBUserClient.kext
      - IOUSBLib.bundle
      - ...

KEXT Example: PlugIns

- Optional
- Can contain
  - Other KEXTs (sub-KEXTs)
  - Device Interface libs (CFBundles)
  - Any other kind of bundle
- Allows driver suite to be delivered as single KEXT
- Only one level of nesting supported

KEXTs as Libraries

- KEXTs may link against other (library) KEXTs
- Dependencies are expressed in OSBundleLibraries dictionary
- Each entry contains a CFBundleIdentifier and CFBundleVersion
- Library KEXTs must provide OSBundleCompatibleVersion property indicating the oldest version with which it is binary compatible

KEXTs as Libraries

- Driver com.you.driver depends on com.you.library version 2.0
- Library com.you.library is installed
    - CFBundleVersion is 3.0
    - OSBundleCompatibleVersion is 1.0
- Since the driver depends on a version within the library's compatible version range, the two are compatible

KEXT Suites

- Sub-KEXTs, KEXT libraries, and KEXT matching can be used to divide one KEXT into a “KEXT Suite”
- This should be done if significant pieces of the KEXT will not be used all the time
  - Init-time code (such as firmware)
  - Product variants allow for mix-and-match drivers
  - Device may be attached by a variety of buses

KEXT Management at Boot Time

- Only drivers with the OSBundleRequired property are considered at boot time
- BootX loads the Extensions.mkext cache if possible, or scans the Extensions folder if not
    - Uses modification date to decide
- Kernel-resident code links and executes each KEXT as needed

KEXT Management at Run Time

- KEXT daemon (kextd) is launched shortly after multiuser startup by the /etc/rc script
- kextd contacts the kernel and takes over KEXT management duties
- Kernel-resident linking code is jettisoned to free up memory
- kextd processes KEXT load requests from the kernel

What Causes KEXTs to Load?

- KEXTs are loaded only on demand
    - I/O Kit Drivers and Families are requested via family matching as hardware is discovered
    - Filesystem KEXTs are loaded during volume mounting by exec'ing kextload
    - NKEs are loaded by Startup Items, or by particular actions like ppp connection
- The exact rules for mapping an event to a particular KEXT vary by subsystem

What Causes KEXTs to Unload?

- KEXTs are unloaded only on demand
    - I/O Kit Drivers and Families unload about a minute after they are no longer referenced
    - Filesystem KEXTs can unload during volume unmount
    - NKEs typically do not unload, or are unloaded due to particular actions, such as ppp connection tear-down
- The kextunload utility will attempt to unload a KEXT, though it may fail the unload request

KEXT Binary Compatibility Issues

- KEXT management is based on static linking—we are limited to the information provided by the compiler in the KEXT's symbol table
- List all CFBundleIdentifiers on which you depend, using the oldest version number that supports the APIs you need (see "Kernel Extension Dependencies")
- Build using the headers from the oldest version of the OS on which you want to run
- Use gcc 2.95 if you want to run on anything prior to Jaguar

KEXT Binary Compatibility Issues

- Avoid direct use of Mach and BSD APIs if possible; they are more likely to change in binary-incompatible ways
- We will be working to create supportable APIs in the future for network and filesystem KEXTs
- Developers will need to migrate to these new APIs, since we will have to deprecate existing ones to move forward
- Stay in contact with DTS or watch the Darwin groups to find out how these transitions will be staged

Manipulating KEXTs From Applications

- KEXTs should be installed by unpacking to /tmp, then moving to /System/Library/Extensions
- Remove KEXTs by moving them to /tmp
- "touch /System/Library/Extensions" after changing any driver to ensure all KEXT caches are updated
- KEXTs need not be installed in /System/Library/Extensions to be loaded
- KEXTs may be loaded or unloaded by running the KEXT utilities from a setuid utility; use fork( ) and exec( )

KEXT Preferences

- Ideally, avoid KEXT preferences completely
    - Adds complexity to the UI
    - More state to manage (and corrupt)
- Preferences may not be stored in the KEXT bundle
- Store preferences in /Library/Preferences
- Use KEXT's CFBundleIdentifier as the file name

KEXT Preferences

- Provide a prefs utility as a
  - Resource of your KEXT
  - Stand-alone app
  - Preference pane in */Library/PreferencePanes
- The utility may present UI to the user
- Utility may use CFPreferences to read and write the preference file (see "Core Foundation Preference Services")
- Utility may communicate with your driver instances using any available access mechanism

Development Tips

Loading and Unloading

- Add IOKitDebug=65535 (integer) to your driver's personality to get additional logging during matching—look in /var/log/system.log
- Use /usr/sbin/ioreg -c myClass to see where your driver fits into the IORegistry
- Use /usr/sbin/ioclasscount myClass . . to see how many instances of your driver remain—can help diagnosing kextunload failures

Development Tips

Panics and Hangs

- Execute /usr/sbin/nvram boot-args="debug=4" and reboot your test machine to enable remote attach—this allows Cmd+Pwr to interrupt many hangs
- Do not put your kexts in /System/Library/Extensions; load them manually using kextload

Development Tips

Panics and Hangs

- Make use of remote debugging using symbols from manual kextload—see “Hello Debugger” tutorial
- IOLog( ) and printf( ) are not synchronous and have a limited bandwidth. If you want to see all messages leading up to a panic or hang, log in as “>console” before triggering failure

Preparing a KEXT for Deployment

- Make sure code is ready for release:
  - Remove assert( ), Debugger( ) calls . . .
  - Get rid of diagnostic messages
  - Remove IOKitDebug property or set to 0
  - Do not forget to set appropriate version numbers
- Build with “Deployment” build-style in PB, or at least run strip -S all KEXT binaries
- Package the KEXT using a tool such as PackageMaker; See “Packaging Your KEXT for Distribution and Installation”

Preparing a KEXT for Deployment

- Always test your installation process!
- Verify that each KEXT is correctly installed
    - Ownership must be (root:wheel)
    - Directory permissions should be 755
    - File permissions should be 644

Managing Kernel Extensions

Nik Gervae
KEXT Management Engineer

New KEXT Tools and Techniques

- All-new codebase
    - Much more extensible and maintainable
- Built on a comprehensive library
    - In Darwin under IOKitUser and kext_tools projects
    - Not for third party use yet, but someday . . .

The Tools

- kextload (obsoletes kmodload, kmodsyms)
  - Many new options for debugging
  - Installed with base system
  - Usage compatible with prior versions
- kextunload (obsoletes kmodunload)
  - Installed with base system
  - Usage compatible with prior versions

The Tools

- kextstat (obsoletes kmodstat)
    - Do not have to be root to run
    - Installed with developer tools; not in base system
- kextcache (obsoletes mkextcache)
    - Installed with the base system

KEXT Manager Daemon

- kextd adds options
  - Installed with base system, but considered an implementation detail
  - Do not rely on presence of kextd executable or process

Using kextload

- Verbose logging: use -v for 6 levels of verbose logging (WWDC seed contains a bug at level 6, so avoid using it when loading a kext)
- Generating symbols: use -s
    - Automatically generates symbols for all libraries too; no more kmodsyms -d ... -d ...
- Generating symbols without loading: use -n, -a, -A, -k

Using kextload

- Debugging driver start() functions: use -l, -m
- Debugging other KEXT start routines: use -i, -I
- Specifying dependencies explicitly: use -d, -r, -e
- Skipping authentication during development: use -z (DevTools version only)
- kextload performs strict authentication if you use any new option, as well as with -i

Verifying Your KEXT

- Use the -t option to perform a full set of checks on your KEXT
- Diagnoses problems with the info dictionary, with file ownership and permissions, and with library dependencies
- Always run kextload -nt on your KEXT as a test before shipping software

Creating Multi-KEXT Caches

- kextcache replaces mkextcache
- Allows precise inclusion of KEXTs with -l, -L, -n, -N options
- With -l and -n, KEXTs named on the command line are always included regardless of OSBundleRequired setting
- With -L and -N, all KEXTs are screened by OSBundleRequired
- One small bug in WWDC seed: PlugIns missed on explicitly-named KEXTs

Examining Loaded KEXTs

- kextstat replaces kmodstat
- Does not require running as root
- Skip in-kernel components with -k
- Skip header with -l; useful for shell script processing
- Get info about a specific KEXT with -b

Finding a KEXT

- New function in IOKitUser finds KEXTs installed in /System/Library/Extensions

```
CFURLRef
    KextManagerCreateURLForBundleIdentifier(
        CFAllocatorRef allocator,
        CFStringRef bundleIdentifier);
```

KEXT Tools Suite History

- Mac OS X 10.0
  - Boot-time driver loading support
- Mac OS X 10.1
  - Extension.mkext cache introduced to improve boot performance
- Jaguar
  - Update toolset
  - Build base for new features
  - Improved performance and footprint

Future Directions

- Parity between kernel and user-space code
- Clean up kernel “kmod” API; avoid using
- Make KEXT management library available for apps
- KEXT bundle signing
- Input from developers

Roadmap

100 The Darwin Road Map	Room A1 **Mon., 2:00pm**
103 Open Source, Apple, and You	Civic **Tue., 2:00pm**
107 The Darwin Kernel	Civic **Wed., 9:00am**
FF002 Darwin	Room J1 **Wed., 3:30pm**

Who to Contact

Craig Keithley
USB and FireWire Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2002/urls.html

For More Information

- Apple Developer Website
  http://developer.apple.com
- Darwin Project Website and Mailgroups
  http://developer.apple.com/darwin/

Documentation

KEXT Management

- Man pages
  - kextload, kextunload, kextcache, kextstat, kextd
- Release note
- Revised I/O Kit documentation

Documentation > Darwin > I/O Kit Documentation

http://developer.apple.com/techpubs/macosx/Darwin/index.html

Q&A

```
/desktop: No such file or
[duarte:~] ryan% ls
Desktop     Library     Mus
Documents   Movies      Pic
[duarte:~] ryan% cd /Libr
[duarte:/Library] ryan% c
[duarte:/Library/Webserve
CGI-Executables  Document
```

Craig Keithley
USB and FireWire Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2002/urls.html

WWDC2002

WWDC2002

WWDC2002